



**Eur päisches  
Patentamt**

**Eur pean  
Patent Office**

**Office eur péen  
des brevets**

**Bescheinigung**

**Certificate**

**Attestation**

Die angehefteten Unterla-  
gen stimmen mit der  
ursprünglich eingereichten  
Fassung der auf dem näch-  
sten Blatt bezeichneten  
europäischen Patentanmel-  
dung überein.

The attached documents  
are exact copies of the  
European patent application  
described on the following  
page, as originally filed.

Les documents fixés à  
cette attestation sont  
conformes à la version  
initialement déposée de  
la demande de brevet  
européen spécifiée à la  
page suivante.

**Patentanmeldung Nr.    Patent application No.    Demande de brevet n°**

02368105.9

Der Präsident des Europäischen Patentamts;  
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets  
p.o.

**R C van Dijk**



information (programs and data) is typically stored on the hard-disk and loaded (at least partially) into the working memory when the programs are running. The programs are initially installed onto the hard disk from CD-ROM.

5       An operating system (OS) 205 (such as Windows XP) provides a software platform for the computer, on top of which other programs can run. A Java Runtime Environment (JRE) 210 is installed over the operating system 205. The JRE 210 consists of a Java Virtual Machine (JVM), Java core classes and supporting files, which define a standard platform for  
10       running applications written in the Java language; particularly, the JVM provides an abstract computing machine for Java applications, which machine is independent of the underlying (hardware and software) platform of the computer.

15       The computer implements an infrastructure 212 for managing several kinds of physical or logical resources; for example, the resources are stand-alone programs, client-server applications, INTERNET-based services, computers, hardware devices, controllers, people, organizations, and the like.  
20       Each resource is instrumented to enable access to corresponding information (such as data, events and status). For this purpose, the resource is modeled as an object conforming to the CIM specification. The CIM model defines a standard schema for data with a basic set of classes, methods  
25       and associations; particularly, core models incorporate notions applicable to all management domains, whereas common models incorporate notions applicable to specific management domains (such as networks, applications, devices and users).

      The structure and the content of the models are described  
30       in the Managed Object Format (MOF). The MOF syntax supports keywords (known as qualifiers), which are enclosed between brackets before a class or property declaration; each qualifier consists of a name/value pair that describes and characterizes the class or property. A particular qualifier  
35       "Key" is used to reference managed objects consisting of specific instances of a CIM class; particularly, the managed

objects are named by concatenating the values of all the properties of the class that are qualified by "Key" (which concatenation defines a so-called modelname).

The CIM model is supported by the Windows Management Instrumentation (WMI) architecture. The WMI architecture includes a framework 215 (known as CIM Object Manager, or CIMOM), which provides a uniform way for management applications (or WMI clients) 220 to access the managed objects (irrespective of their implementation). For this purpose, the CIMOM 215 exposes an interface "IWbemServices", which defines a series of methods that can be called by the WMI clients 220. For example, the CIMOM 215 allows managed objects to be created, deleted, and manipulated; moreover, it supports event notification and query-based data retrieval.

The CIMOM 215 interfaces with a WMI repository 225. The repository 225 stores static management data (managed objects and associations). Management data in the repository 225 is organized in logic collections (schema), which are loaded into corresponding namespaces; a namespace provides a domain (or container) for a schema and for the instances of the CIM classes in that schema. The namespace path and its modelname unambiguously specify each managed object in the repository 225.

The CIMOM 215 further exploits one or more WMI providers 230 of dynamic management services. Each instance of a WMI provider 230 acts as an intermediary between the CIMOM 215 and associated managed objects 235; the instance of the WMI provider 230 always supports a single namespace (which is set during its initialization). The WMI provider 230 manages dynamic data relating to the managed objects 235 (which data is not available in the repository 225); in addition, the WMI provider 230 implements notification of events that are not supported by the CIMOM 215. Information about which WMI provider 230 to contact for dynamic data or event notifications is stored in the repository 225. Particularly, a CIM class whose instances are created dynamically is denoted



Anmeldung Nr:  
Application no.: 02368105.9  
Demande no:

Anmeldetag:  
Date of filing: 27.09.02  
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

INTERNATIONAL BUSINESS MACHINES CORPORATION

Armonk, NY 10504  
ETATS-UNIS D'AMERIQUE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:  
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.  
If no title is shown please refer to the description.  
Si aucun titre n'est indiqué se référer à la description.)

A method and system for integrating non-compliant providers of dynamic services  
into a resource management infrastructure

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s)  
revendiquée(s)

Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/  
Classification internationale des brevets:

G06F9/40

Am Anmeldetag benannte Vertragsstaaten/Contracting states designated at date of  
filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR IE IT LI LU MC NL PT SE SK TR



**A METHOD AND SYSTEM FOR INTEGRATING NON-COMPLIANT PROVIDERS  
OF DYNAMIC SERVICES INTO A RESOURCE MANAGEMENT  
INFRASTRUCTURE**

**Technical field**

5       The present invention relates to the data processing field, and more specifically to a method and system for integrating non-compliant providers of dynamic services into a resource management infrastructure.

**Background art**

10

Management of entities representing different types of resources (such as software applications or hardware devices) is a critical issue in a data processing system, especially a computer network. This problem is particular acute for  
15 resources that have a high level of complexity or are distributed across a large number of installations.

Several technologies have been proposed in the last years for implementing a resource management infrastructure. For example, the Common Interface Model (CIM) is a standard

specification that abstracts and describes the resources to be managed using an object-oriented paradigm. The CIM model may be supported by different frameworks, which provide a uniform way for management applications to access the managed objects.

5 Each framework (also known as CIM Object Manager, or CIMOM) exploits providers of dynamic management services (each one acting as an intermediary between the framework and associated managed objects).

10 Particularly, the Windows Management Instrumentation (WMI) is the implementation of the CIM model proposed by Microsoft Corporation and running on the Windows operating system. In the WMI architecture, the CIM providers must be implemented as objects conforming to the Common Object Model (COM) specification and must comply with a particular WMI  
15 provider interface (named "IWbemServices").

The COM objects are natively supported only on the Windows operating system. This is a major problem, which strongly limits the integration of non-compliant providers available for different operative environments into the WMI  
20 architecture. Moreover, this limits the ability to implement providers that can be used with frameworks running on different operating systems.

### **Summary of the invention**

25 It is an object of the present invention to integrate non-compliant providers of dynamic services into the resource management infrastructure.

It is another object of the present invention to achieve this integration without substantially affecting operation of  
30 the resource management infrastructure.

It is yet another object of the present invention to integrate the non-compliant providers into the resource management infrastructure in a simple and effective way.



The accomplishment of these and other related objects is achieved, in a resource management infrastructure including a framework for allowing management applications to access managed objects, the framework supporting compliant providers  
5 of dynamic services for the managed objects running in a first operative environment, by a method of integrating a plurality of non-compliant providers of dynamic services for a set of the managed objects, the non-compliant providers running in a second operative environment that is not supported by the  
10 framework, wherein the method includes the steps of: the framework requesting a first dynamic service for a selected managed object of the set to a compliant provider associated with the selected managed object, the compliant provider identifying a non-compliant provider associated with the  
15 selected managed object, the compliant provider requesting a second dynamic service for the selected managed object, corresponding to the first dynamic service, to an intermediate entity running in the second operative environment passing an indication of the non-compliant provider, the intermediate  
20 entity routing the request of the second dynamic service to the non-compliant provider.

Moreover, the present invention also provides a computer program for performing the method and a product storing the program.

25 The invention also encompasses a corresponding system for integrating the non-compliant providers, and a resource management infrastructure including this system.

The novel features believed to be characteristic of this invention are set forth in the appended claims. The invention  
30 itself, however, as well as these and other related objects and advantages thereof, will be best understood by reference to the following detailed description to be read in conjunction with the accompanying drawings.

#### **Brief description of the drawings**

- Figure 1** is a schematic block diagram of a computer in which the method of the invention can be used;
- Figure 2** shows a partial content of a working memory of the computer;
- Figures 3a and 3b** illustrate an activity diagram of a resource management process;
- Figures 4a-4b** are flow charts describing how different time-outs are managed in the process.

#### **Detailed description of the preferred embodiment**

With reference in particular to Figure 1, a data processing system 100 (such as a Personal Computer) is shown.

5 The computer 100 is formed by several units, which are connected in parallel to a communication bus 105. In detail, a microprocessor ( $\mu$ P) 110 controls operation of the computer 100, a Read Only Memory (ROM) 115 stores basic code for a bootstrap of the computer 100, and a DRAM 120 is directly used

10 as a working memory by the microprocessor 110. Several peripheral units are further connected to the bus 105 (by means of respective interfaces). Particularly, a mass memory consists of a magnetic hard-disk 125 and a driver 130 for reading CD-ROMs 135. Moreover, the computer 100 includes input

15 devices 140 (for example, a keyboard and a mouse), and output devices 145 (for example, a monitor and a printer). A network Interface Card (NIC) 150 is used to connect the computer 100 to a network, typically the INTERNET.

Similar considerations apply if the computer has a

20 different structure (for example, with a multiprocessor architecture), if the computer includes other units (such as a scanner), if the computer is replaced with an equivalent data processing system (for example, consisting of a network of computers), and the like.

25 Considering now Figure 2, a partial content of the working memory 120 of the computer in operation is shown; the

preceding its declaration with a qualifier "Dynamic" and a  
qualifier "Provider"; the value of the qualifier "Provider" is  
the name of the WMI Provider 230 associated with all the  
managed objects instantiating the CIM class. Different WMI  
5 providers 230 may be specified before selected properties; in  
this case, the property is supported by the corresponding WMI  
provider 230 (instead of the one defined at class level).

The WMI providers 230 must be objects conforming to the  
COM specification, which objects implement the interface  
10 "IWBemServices". The WMI providers 230 may be either in-proc  
or out-proc; in the first case, the WMI providers are  
implemented as native Windows libraries (DLL) and run in the  
same process of the CIMOM 215, whereas in the second case they  
are stand-alone executables. An additional (out-proc) WMI  
15 provider 240 acts as a bridge between the WMI architecture and  
the Java environment. The bridge provider 240 communicates  
with a Java intermediate object 245 through the Java Native  
Interface (JNI) 255. The JNI 255 allows Java objects to  
operate with native applications and libraries written in  
20 other languages; vice-versa, the native methods can utilize  
Java objects in the same way that Java code uses these  
objects. As described in detail in the following, the  
intermediate object 245 is used to create a JVM instance 257  
and to launch a series of providers 260 written in the Java  
25 language. The Java providers 260 support dynamic services for  
a set of corresponding managed objects (denoted with 265). The  
name of the Java provider 260 associated with a specific CIM  
class or property is indicated in the MOF description by means  
of a new qualifier "Java\_Provider" (preceding its  
30 declaration). This also allows for simple and even dynamic  
assignment of Java providers 260 to a resource modeled with a  
particular CIM class, thereby maintaining the loose-coupling  
of managed objects and providers typical of the CIM model.

Similar considerations apply if the programs are  
35 distributed on any other computer readable medium (such as one  
or more floppy-disks), if different modules or functions are

provided, if an equivalent resource management infrastructure is implemented (even based on a different model), if the CIM models are expressed with another syntax (for example, using the Extensible Markup Language, or XML), if all the managed  
5 objects are associated with Java providers, and the like. Alternatively, the framework runs in a different operative environment (such as UNIX), the non-compliant providers are written in another language (for example, the PERL or a shell language), the bridge provider is in-proc, the bridge provider  
10 interfaces with the intermediate object (or an equivalent entity) in another way, or the bridge provider and the Java providers are associated with the managed objects in a different manner.

Moving now to Figures 3a and 3b, the flow of activities  
15 of a generic resource management process 300 is illustrated. Whenever a WMI client needs to perform some operations on the managed objects, a corresponding method exposed by the interface "IWbemServices" of the CIMOM is called. If the operation only involves static management data that is hold in  
20 the repository, the CIMOM processes the request directly. Conversely, the relevant CIM class in the repository is inspected to identify the associated WMI providers to be exploited; one or more methods are then called on each WMI provider for implementing the requested service.

25 Assuming that the request relates to a CIM class specifying the bridge provider with the qualifier "Provider", the process begins at the black start circle 303 wherein the CIMOM calls the method on the bridge provider (passing the relevant CIM class as a parameter). In response thereto, the  
30 bridge provider in turn inspects the CIM class at block 306, so as to identify the corresponding Java provider (specified with the qualifier "Java\_Provider").

A semaphore is then locked at block 309; the semaphore is used to ensure that operations relating to the creation of the  
35 JVM are carried out in mutual exclusion (among concurrent threads running the bridge provider). A test is made at

decision block 312 to determine whether the JVM has already been created; for this purpose, the bridge provider verifies whether a corresponding global pointer is set. If so, the process descends into block 315 (described in the following).

5 Conversely, the bridge provider at block 318 creates the JVM and sets the corresponding global pointer (using the JNI). Continuing to block 321, the intermediate object is instantiated inside the JVM. The process then passes to block 315. Considering now block 315, the semaphore on the JVM is  
10 unlocked (thereby resuming the threads waiting on it).

The process proceeds to block 324, wherein the information passed to the bridge provider in the called method is converted from MOF types to Java types. More specifically, the bridge provider converts all the parameters (if any)  
15 passed with the called method. Moreover, each method of the interface "IWbemServices" may include an additional parameter pointing to an object of the type "IWbemContext". This object (consisting of a simple container of named values) is used by the WMI clients to communicate context information to the WMI  
20 providers, which information cannot be specified in the standard parameters; the information is completely opaque to the CIMOM, and it is simply forwarded to the WMI providers. If the pointer is not null, the bridge provider enumerates and converts every value included in the object "IWbemContext".

25 In both cases, the conversion is carried out simply assigning a pre-set Java type to each MOF type, as specified in the following table:

MOF type	Windows Automation (VARIANT) type	Java type
Boolean	VT_BOOL	java.lang.Boolean   boolean
Datetime	VT_BSTR	Date
<Classname>   object   ref	VT_UNKNOWN	Non-applicable, since parameters of this type contain references to instances of different classes that are handled directly by CIMOM and must be ignored by the WMI providers
Void	-	java.lang.Void   void
Sint8	VT_I2	java.lang.Byte   byte
Sint16	VT_I2	java.lang.Integer   int
Sint32	VT_I4	java.lang.Integer   int
Sint64	VT_BSTR	java.lang.Long   long
Real32	VT_R4	java.lang.Float   float
Real64	VT_R8	java.lang.Double   double
Uint8	VT_UI1	java.lang.Integer   int
Uint16	VT_I4	java.lang.Integer   int
Uint32	VT_I4	java.lang.Integer   int

Uint64	VT_BSTR	java.lang.Long   long
Char16	VT_I2	java.lang.String
String	VT_BSTR	java.lang.String

With reference now to block 327, a method corresponding to the one called on the bridge provider is called on the intermediate object (using the JNI); the identifier of the managed object (namespace path and modelname), the name of the Java provider and the converted parameters are passed to this method. In response thereto, the intermediate object locks a further semaphore at block 330, in order to ensure that the operations relating to the instantiation of the Java provider are carried out in mutual exclusion (among concurrent threads running the intermediate object). A test is made at decision block 333 to determine whether the Java provider has already been instantiated; for this purpose, the intermediate object verifies whether an identifier of the Java provider is stored in a global object "Provider\_table" of the type "Hashtable". If not, the process continues to block 336; the intermediate object instantiates the Java provider and updates the object "Provider\_table" accordingly. The process then continues to block 339. Conversely (Java provider already instantiated), the flow of activities descends into block 339 directly. Considering now block 339, the semaphore on the Java provider is unlocked.

Moving to block 342, a method corresponding to the one called on the intermediate object is then called on the Java provider, passing the identifier of the managed object and the converted parameters. The process continues to block 345 in the swim-lane of the Java provider, wherein the operations needed to implement the requested service are carried out. Any results and/or exceptions of the method are returned to the intermediate object at block 348.

Referring back to the swim-lane of the intermediate object, an expiration time of the Java provider is calculated at block 351 by adding a predetermined value (for example,

0.5h) to a current time. A test is made at decision block 354 to verify whether a record specifying the Java provider is present in a global FIFO queue (used to manage a time-out for all the Java providers, as described in the following). If so, the record is removed from the FIFO queue at block 357, and the process then continues to block 360; conversely, the flow of activities descends into block 360 directly. Considering now block 360, a new record including an identifier of the Java provider together with its expiration time is inserted into the FIFO queue. A timer is then reset at block 363 to the expiration time of the first record to be extracted from the FIFO queue.

The process proceeds to block 366, wherein the results and exceptions (if any) received from the method called on the Java provider are returned to the bridge provider (using the JNI). Referring now to block 369 in the swim-lane of the bridge provider, a further timer (used to manage a time-out for the JVM, as described in the following) is reset to a predetermined value (for example, 0.5h). Proceeding to block 372, the results collected from the method called on the Java provider are converted back from Java types to MOF types; at the same time, any caught exception is mapped to a corresponding error code (for example, exploiting a pre-set table). An appropriate result of the method originally called on the bridge provider is constructed using the converted results and exceptions, and it is then returned to the CIMOM at block 375 (in order to allow the CIMOM to complete the operations requested by the WMI client). The process then ends at the concentric white/black stop circles 378.

Moving now to Figure 4a, the operations carried out by the bridge provider for managing the time-out for the JVM implements a method 400a. The method starts at block 403, and then passes to block 406 in response to a signal received from the corresponding timer at the end of the time interval to which it was reset (at block 369 in Figure 3b). A test is made at decision block 409 to determine whether the JVM is



currently in use (by other threads). If not, the JVM is destroyed at block 412, and the method ends at the final block 415. Conversely, the method descends into the final block 415 directly.

5        Likewise, the operations carried out by the intermediate object for managing the time-out for the Java providers are illustrated in Figure 4b. The operations implement a method 400b that starts at block 453. The method then passes to block 456 in response to a signal received from the corresponding  
10 timer at the end of the time interval to which it was reset (at block 363 in Figure 3b). A test is made at decision block 459 to determine whether the Java provider specified in the first record to be extracted from the FIFO queue is currently in use. If not, this Java provider is destroyed at block 462  
15 and the corresponding record is removed from the FIFO queue at block 465; the method then passes to block 468. Conversely, the method descends into block 468 directly. With reference now to block 468, the timer is reset to the expiration time of the new first record to be extracted from the FIFO queue. The  
20 method then ends at the final block 471.

For example, let us consider a generic CIM class "MyClass" defined by the following file "MyClass.mof":

```

25      [Dynamic,
        Description ("MyClass description"),
        Provider("BridgeProvider"),
        Java_Provider("mypackage.MyJavaProvider")
      ]
      class MyClass
      {
30          [Key,
            Description ("MyClass key identifying any instance thereof")
          ]
          sint32 InstanceIdentifier;
          [Description ("Some properties associated with the Java provider
35          defined at class level")]
          String MyPropertyA;
          String MyPropertyB;
          String MyPropertyC;

```

```

        [Description ("A property associated with a different Java
        provider"),
        Java_Provider("mypackage.MyJavaProvider1")
        ]
5      String MyProperty1;
    }

```

Typically, all the operations requested to the CIMOM by the WMI clients are carried out calling a method "CreateInstanceEnumAsync" on the appropriate providers. This method creates an enumerator that asynchronously supplies the instances of a CIM class according to specified selection criteria; the method returns immediately after being called so that the CIMOM can continue its execution while the request is pending. The signature of the method "CreateInstanceEnumAsync" is:

```

15      HRESULT CreateInstanceEnumAsync(
            const BSTR srtClass,
            LONG lFalgs,
            IWbemContext* pCtx,
20      IWbemObjectSink* pResponseHandler
        )

```

The parameter "srtClass" includes the name of the CIM class for which instances are desired, the parameter "lFlags" is a flag defining operation of the method, the parameter "pCtx" is a pointer to the object "IWbemContext", and the parameter "pResponseHandler" is a pointer to the object receiving the instances of the CIM class as they become available; the method returns a value indicating the status of the operation.

After the instances of the CIM class have been enumerated, the CIMOM can perform different operations. For example, a method "GetProperty" retrieves the value of an individual property, a method "PutProperty" updates the value of a selected property, and a method "ExecMethod" executes a method exported by the WMI provider.

Accordingly, each Java provider implements an interface "JavaProviderInterface", which exposes methods corresponding to the ones of the bridge provider. This interface is defined as follows:

```

import java.util.*;
public interface JavaProviderInterface
{
    5      public Enumeration createInstanceEnum(String nameSpace, String
          className, Hashtable context)
          throws CIMException;

    public Object getProperty(String nameSpace, String className,
    10      Hashtable targetInstance, String propertyName, Hashtable
          context)
          throws CIMException;

    public Hashtable getMultipleProperties(String nameSpace, String
    15      className, Hashtable targetInstance, String propertyNames[],
          Hashtable context)
          throws CIMException;

    public void putProperty(String nameSpace, String className,
    20      Hashtable targetInstance, String propertyName, Object
          propertyValue, Hashtable context)
          throws CIMException;

    public Object execMethod (String nameSpace, String className,
    25      Hashtable targetInstance, String methodName, Vector inParms,
          Vector outParms, Hashtable context)
          throws CIMException;
}

```

25 In detail, the method createInstanceEnum receives a parameter "nameSpace" including the namespace path of the CIM class, a parameter "className" including the name of the CIM class, and a parameter "context" including the context information passed to the bridge provider; the method throws exceptions in the object "CIMException" and returns an enumeration of objects "Hashtable", each one for a retrieved instance of the CIM class.

35 The method "getProperty" receives the parameters "nameSpace", "className" and "context", a parameter "targetInstance" including the modelname of the managed object, and a parameter "propertyName" including the name of the selected property; the method throws the exceptions "CIMException" and returns the value of the property. The

additional method "getMultipleProperties" receives the parameters "nameSpace", "className", "context", "targetInstance", and a parameter "propertyName[]" including a list of the names of the properties associated with the Java provider defined at class level; the method throws the exceptions "CIMException" and returns the values of all these properties at the same time. The method "putProperty" receives the parameters "nameSpace", "className", "targetInstance", "propertyName" and "context", and a parameter "propertyValue" including the value to be assigned to the selected property; the method throws the exceptions "CIMException" and does not return any value. At the end, the method "execMethod" receives the parameters "nameSpace", "className", "targetInstance" and "context", a parameter "methodName" including the name of the method to be called on the managed object, a parameter "inParms" including the values to be passed to the called method, and a parameter "outParms" for the values returned by the called method; the method throws the exceptions "CIMException" and returns a result of the operation.

For example, let us consider that the CIMOM needs to enumerate the instances of the CIM class "MyClass". For this purpose, the CIMOM calls the method "createInstanceEnumAsync" on the associated WMI provider "BridgeProvider". In response thereto, the WMI provider "BridgeProvider" inspects the CIM class "MyClass", so as to identify the corresponding Java provider "MyJavaProvider". The method "createInstanceEnum" is then called on the intermediate object, passing the information received from the CIMOM (after their conversion) together with the string "MyJavaProvider". The intermediate object in turn routes this method to the Java provider "MyJavaProvider". The method returns an enumeration of the retrieved instances of the CIM class "MyClass" to the intermediate object, which passes the enumeration back to the WMI provider "BridgeProvider". For each instance, the WMI provider "BridgeProvider" creates a CIM instance, which is used to store information about the managed object; an entry

of the CIM instance corresponding to the property "Key" is filled with the name of the instance (after its conversion).

The CIM class "MyClass" is then inspected to identify its properties (and their associated Java providers). The method  
5 "getMultipleProperties" is then called on the intermediate object and then on the Java provider "MyJavaProvider", passing the names of all the properties associated with the Java provider defined at class level (i.e., "PropertyA", "PropertyB" and "PropertyC" in the example at issue). The  
10 values of these properties are returned to the intermediate object and then to the WMI provider "BridgeProvider"; the values are then stored in the corresponding entries of the CIM instance (after their conversion). For each property associated with a different Java provider, the method  
15 "getProperty" is then called on the associated Java provider through the intermediate object (in the example at issue, the method "getProperty" is called on the Java provider "MyJavaProvider1" passing the string "MyProperty1"). The value of the property returned by the Java provider through the  
20 intermediate object is then stored in the corresponding entry of the CIM instance (after its conversion). As soon as the values of all the properties have been retrieved, the CIM instance is returned to the CIMOM.

Similar considerations apply if equivalent methods are  
25 performed, if the operations relating to the creation/destruction of the JVM (or of an alternative operative environment) are managed in a different way (for example, with the operative environment that is removed immediately after its use), if the non-compliant providers are  
30 created/destroyed in another manner, if the information passed by and returned to the bridge provider is converted with a different mechanism, if context information is not supported, if the time-outs have different values, if the time-outs are configurable dynamically, if the exceptions thrown by the  
35 methods called on the Java providers are used in another way, and the like.

More generally, the present invention proposes a method for use in a resource management infrastructure. The infrastructure includes a framework for allowing management applications to access managed objects; the framework supports  
5 compliant providers of dynamic services for the managed objects running in a first operative environment. The method of the invention is used to integrate a plurality of non-compliant providers of dynamic services for a set of the managed objects; the non-compliant providers run in a second  
10 operative environment that is not supported by the framework. In the proposed solution, the framework requests a first dynamic service for a selected managed object of the set to a compliant provider associated with the selected managed object. The compliant provider identifies a non-compliant  
15 provider associated with the selected managed object. The compliant provider then requests a second dynamic service for the selected managed object (corresponding to the first dynamic service) to an intermediate entity running in the second operative environment; an indication of the  
20 non-compliant provider is also passed to the intermediate object. In response thereto, the intermediate entity routes the request of the second dynamic service to the non-compliant provider.

The proposed solution allows non-compliant providers to  
25 be integrated in the resource management infrastructure directly; as a consequence, non-compliant providers already available may be used and/or new providers may be developed exploiting any operative environment (even if it is not supported by the framework).

30 This result is achieved without any substantial impact on the operation of the resource management infrastructure. In fact, the bridge provider exposes the usual native interface to the framework; it is then the intermediate object that routes the requests of each dynamic service to the appropriate  
35 provider in a way that is completely opaque to the framework.

The solution of the invention is very simple (since a

single intermediate object acts as a front-end for the bridge provider); moreover, the resulting loss of performance in the resource management framework is substantially insignificant in practice.

5       The preferred embodiment of the invention described above offers further advantages.

10       Particularly, parameters (including context information typical of the WMI architecture) passed to the bridge provider are converted and then passed to the intermediate object (for the Java provider).

A similar mechanism is exploited for returning any result to the framework.

In this way, information can be passed to and/or returned from the Java provider in a very simple way.

15       Preferably, the bridge provider is common to all the CIM classes associated with the Java providers.

The proposed feature concentrates the interface between the WMI architecture and the Java language into a single component.

20       However, the solution according to the present invention leads itself to be implemented even managing the parameters and the results in a different manner, with methods that do not require any parameter and/or do not return any result, or even exploiting two or more bridge providers (for example, 25 interfacing with different categories of Java providers).

Advantageously, the JVM is created by the bridge provider on its first use and it is then re-used for subsequent operations; the JVM is then destroyed after a predetermined time-out.

30       Therefore, processing time is saved since a single JVM is running at any time; moreover, the destruction of the JVM avoids any wastage of system resources when the JVM is not used for a long time.

35       As a further improvement, each Java provider is created and destroyed in a similar manner.

Therefore, the Java providers use system resources only when they are necessary.

Preferably, the time-out for the different Java providers is managed through a single timer (with the aid of a FIFO queue).

The devised mechanism is very simple, but at the same time effective.

However, the proposed solution is also suitable to be implemented creating the JVM and each Java provider in a different way, without destroying the JVM and/or the Java providers, or using a different mechanism for managing the time-out for the Java providers (even with multiple timers).

Advantageously, the solution according to the present invention is implemented with a computer program, which is provided as a corresponding product stored on a suitable medium.

Alternatively, the program is pre-loaded onto the hard-disk, is sent to the computer through a network (typically the INTERNET), is broadcast, or more generally is provided in any other form directly loadable into a working memory of the computer. However, the method according to the present invention leads itself to be carried out even with a hardware structure (for example, integrated in a chip of semiconductor material).

Naturally, in order to satisfy local and specific requirements, a person skilled in the art may apply to the solution described above many modifications and alterations all of which, however, are included within the scope of protection of the invention as defined by the following claims.



## CLAIMS

1. In a resource management infrastructure (212) including a framework (215) for allowing management applications (220) to access managed objects (235,265), the framework supporting  
5 compliant providers (230,240) of dynamic services for the managed objects running in a first operative environment (205), a method (300) of integrating a plurality of non-compliant providers (260) of dynamic services for a set of the managed objects (265), the non-compliant providers running in a second  
10 operative environment (210) that is not supported by the framework, wherein the method includes the steps of:

the framework requesting (303) a first dynamic service for a selected managed object of the set to a compliant provider associated with the selected managed object,

15 the compliant provider identifying (306) a non-compliant provider associated with the selected managed object,

the compliant provider requesting (324-327) a second dynamic service for the selected managed object, corresponding to the first dynamic service, to an intermediate entity running  
20 in the second operative environment passing an indication of the non-compliant provider,

the intermediate entity routing (342) the request of the second dynamic service to the non-compliant provider.

2. The method (300) according to claim 1, further including the  
25 steps of:

the compliant provider converting (324) at least one first parameter received from the framework with the request of the first service into at least one second parameter supported by the second operative environment,

30 the compliant provider passing (327) the at least one second parameter to the intermediate entity with the request of the second dynamic service, and

the intermediate entity passing (342) the at least one second parameter to the non-compliant provider with the request of the second dynamic service.

3. The method (300) according to claim 1 or 2, further including the steps of:

the non-compliant provider returning (348) a result of the second dynamic service to the intermediate entity,

the intermediate entity returning (366) the result of the second dynamic service to the compliant provider,

10 the compliant provider (372) constructing a result of the first dynamic service according to the result of the second dynamic service, and

the compliant provider returning (375) the result of the first dynamic service to the framework.

15 4. The method (300) according to any claim from 1 to 3, wherein the compliant provider is common to all the managed objects of the set.

5. The method (300) according to any claim from 1 to 4, further including the steps under the control of the compliant provider of:

20 verifying (312) whether the second operative environment has been created before requesting the second dynamic service to the intermediate entity,

creating (318) the second operative environment if the result of the verification is negative, and

25 destroying (369,403-415) the second operative environment after a predetermined time-out from a last use thereof.

6. The method (300) according to any claim from 1 to 5, further including the steps under the control of the intermediate entity of:

30

verifying (333) whether each non-compliant provider has been created before requesting the second dynamic service to the non-compliant provider,

5       creating (336) the non-compliant provider if the result of the verification is negative, and

      destroying (354-363,453-471) the non-compliant provider after a predetermined further time-out from a last use thereof.

7. The method (300) according to claim 6, further including the steps under the control of the intermediate entity of:

10       when requesting the second dynamic service to each non-compliant provider, removing (354-357) a possible record including an identifier of the non-compliant provider from a FIFO queue and inserting (360) a new record into the FIFO queue including the identifier of the non-compliant provider together  
15       with an expiration time thereof calculated adding the further time-out to a current time,

      setting (363) a signalling time of a timer to the expiration time of a first record to be extracted from the FIFO queue,

20       in response to the timer reaching the signalling time, destroying (462) the non-compliant provider identified in the first record of the FIFO queue,

      extracting (465) the first record from the FIFO queue, and  
      setting (468) the signalling time of the timer to the  
25       expiration time of the new first record to be extracted from the FIFO queue.

8. A computer program (240,245) directly loadable into a working memory of a computer (100) implementing a resource management infrastructure (212) including a framework (215) for  
30       allowing management applications (220) to access managed objects (235,265), the framework supporting compliant providers (230,240) of dynamic services for the managed objects running in a first operative environment (205), wherein the program when run on the computer performs a method (300) of integrating

a plurality of non-compliant providers (260) of dynamic services for a set of the managed objects (265), the non-compliant providers running in a second operative environment (210) that is not supported by the framework, wherein the method includes the steps of:

a request of a first dynamic service for a selected managed object of the set being received by a compliant provider associated with the selected managed object from the framework,

the compliant provider identifying (306) a non-compliant provider associated with the selected managed object,

the compliant provider requesting (324-327) a second dynamic service for the selected managed object, corresponding to the first dynamic service, to an intermediate entity running in the second operative environment passing an indication of the non-compliant provider,

the intermediate entity routing (342) the request of the second dynamic service to the non-compliant provider.

9. A program product (135) comprising a computer readable medium on which the program (240,245) of claim 8 is stored.

10. In a resource management infrastructure (212) including a framework (215) for allowing management applications (220) to access managed objects (235,265), the framework supporting compliant providers (230,240) of dynamic services for the managed objects running in a first operative environment (205), a system (240,245) for integrating a plurality of non-compliant providers (260) of dynamic services for a set of the managed objects (265), the non-compliant providers running in a second operative environment (210) that is not supported by the framework, wherein the system includes at least one compliant provider (240) associated with the managed objects of the set and an intermediate entity (245) running in the second operative environment, the at least one compliant provider being suitable to receive a request of a first dynamic service

for a selected managed object of the set associated therewith from the framework, being suitable to identify a non-compliant provider associated with the selected managed object, and being suitable to request a second dynamic service for the selected managed object, corresponding to the first dynamic service, to the intermediate entity passing an indication of the non-compliant provider, and the intermediate entity being suitable to route the request of the second dynamic service to the non-compliant provider.

- 10 11. A resource management infrastructure (212) including a framework (215) for allowing management applications (220) to access managed objects (235,265), the framework supporting compliant providers (230,240) of dynamic services for the managed objects running in a first operative environment (205),  
15 a plurality of non-compliant providers (260) of dynamic services for a set of the managed objects (265), the non-compliant providers running in a second operative environment (210) non-supported by the framework, and the system (240,245) according to claim 10 for integrating the  
20 non-compliant providers into the resource management infrastructure.



**A METHOD AND SYSTEM FOR INTEGRATING NON-COMPLIANT PROVIDERS OF  
DYNAMIC SERVICES INTO A RESOURCE MANAGEMENT INFRASTRUCTURE**

**Abstract**

A method and a corresponding system for use in a resource  
5 management infrastructure (212) conforming to the WMI  
architecture are proposed. The WMI architecture includes a  
framework (215), known as CIM Object Manager (or CIMOM), which  
exploits WMI providers of dynamic services for the managed  
objects; in the WMI architecture, the CIM providers must be  
10 implemented as Common Object Model (COM) objects and must  
comply with a specific WMI provider interface (named  
"IWbemServices").

The invention discloses a method for integrating  
non-compliant providers, for example written in the Java  
15 language, into the WMI infrastructure. For this purpose, a  
bridge provider (240) is associated with each managed object  
that is supported by the Java providers. The bridge provider  
identifies the Java provider that must be invoked to fulfill a  
particular request (based on a specific CIM qualifier); the  
20 parameters of the request are converted into Java types, and a  
corresponding method is called on an intermediate Java object  
(245). The intermediate Java object then routes the request to  
the appropriate Java provider.

FIGURE 2





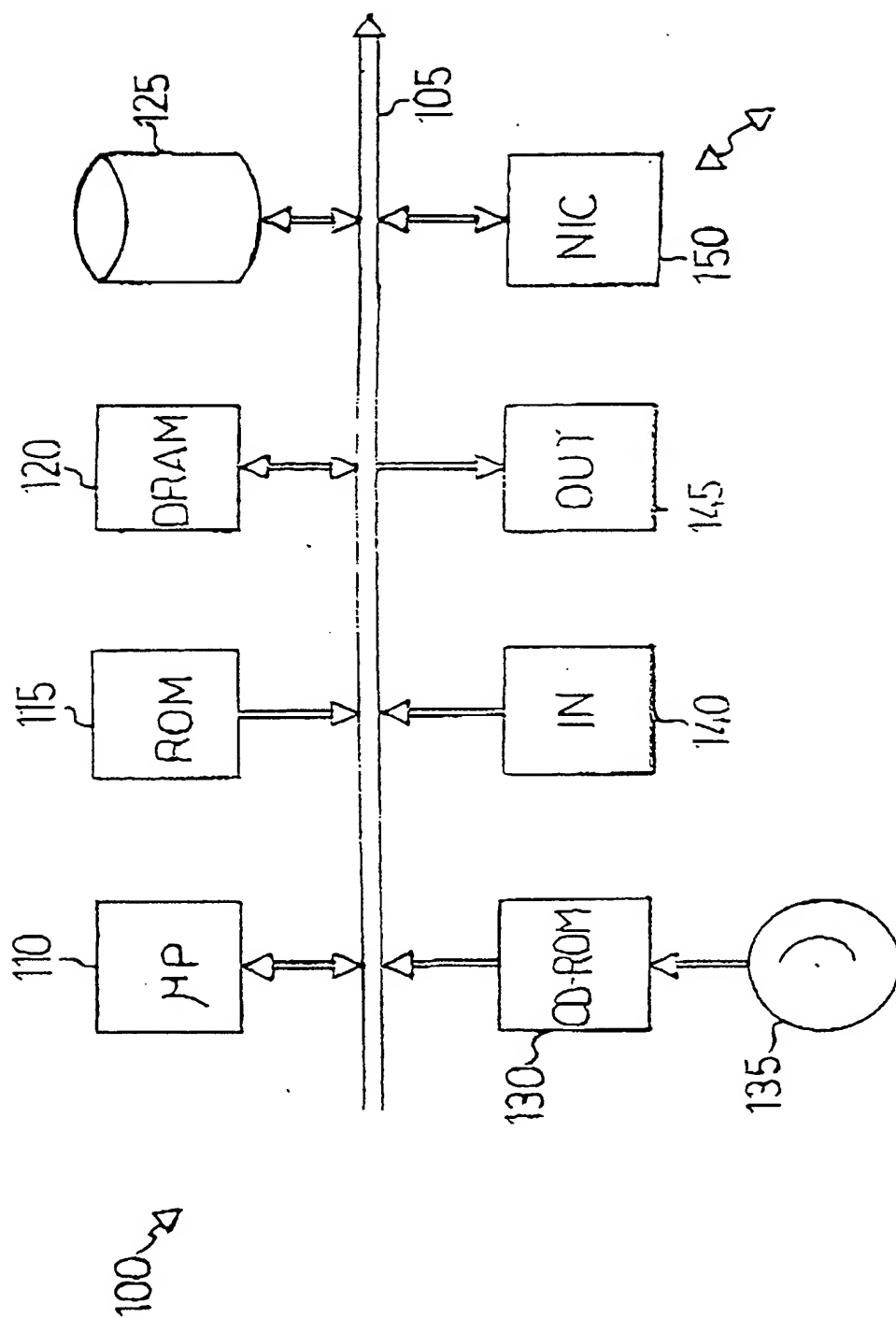


Fig.1

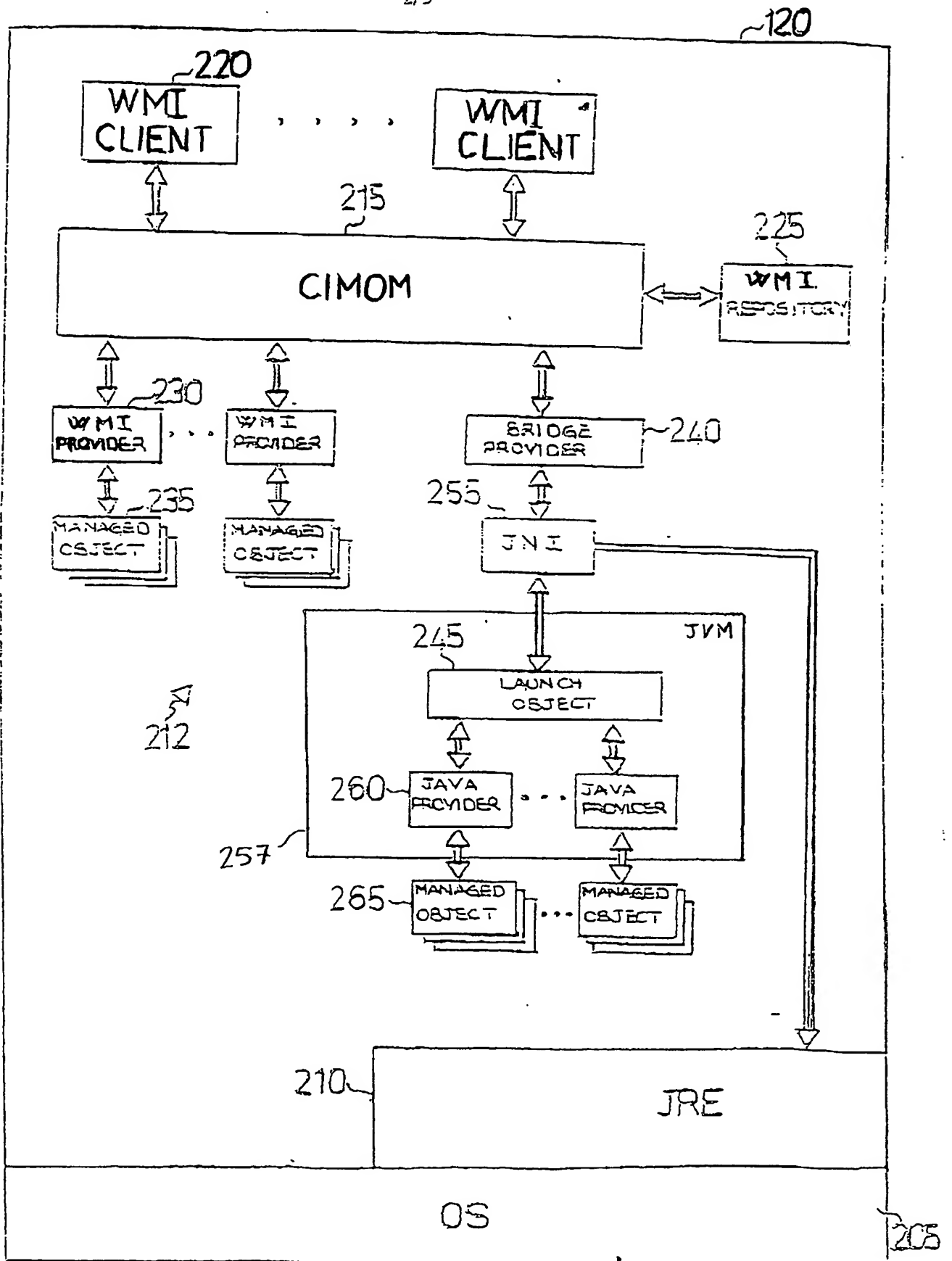


Fig 7

300

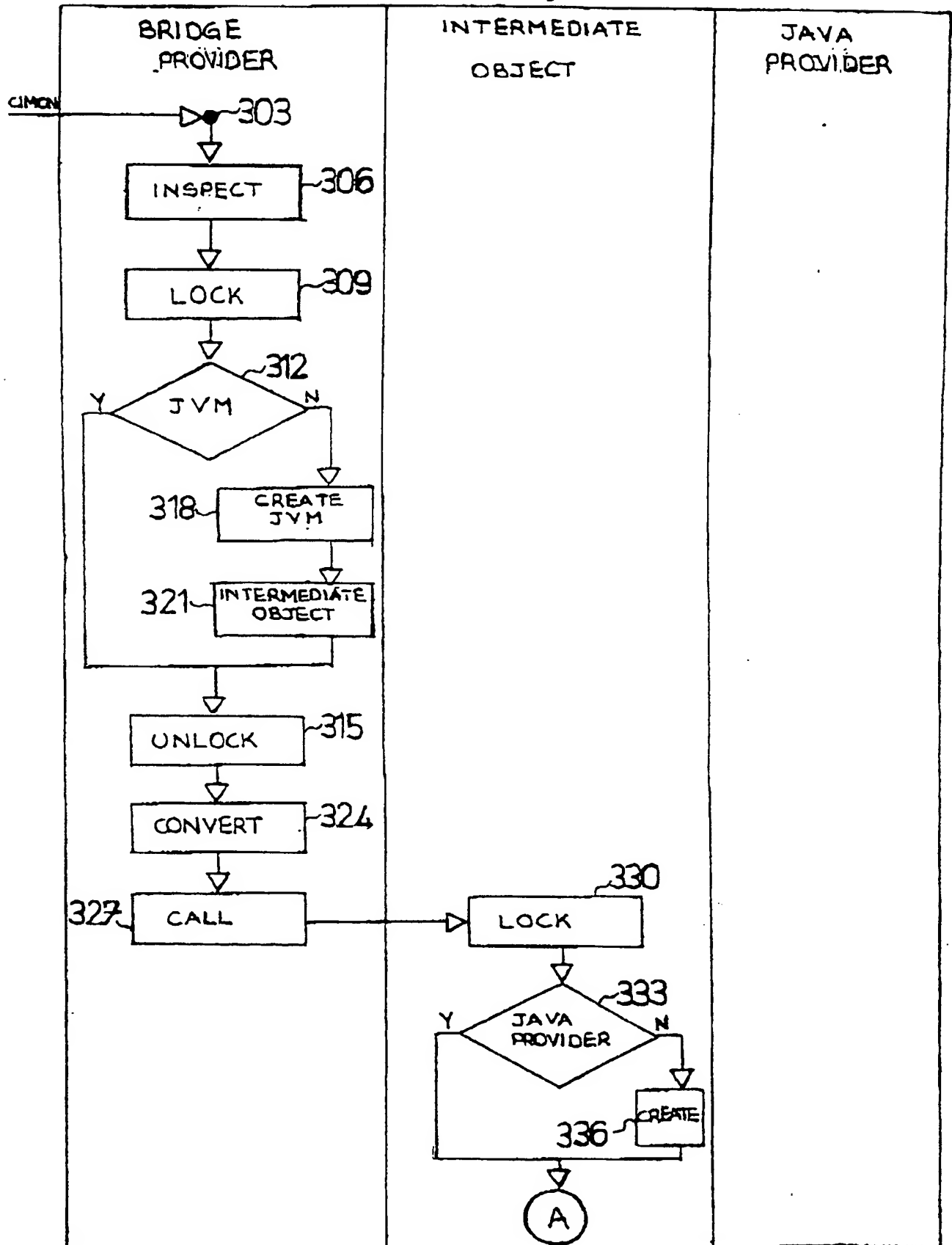


Fig. 3a

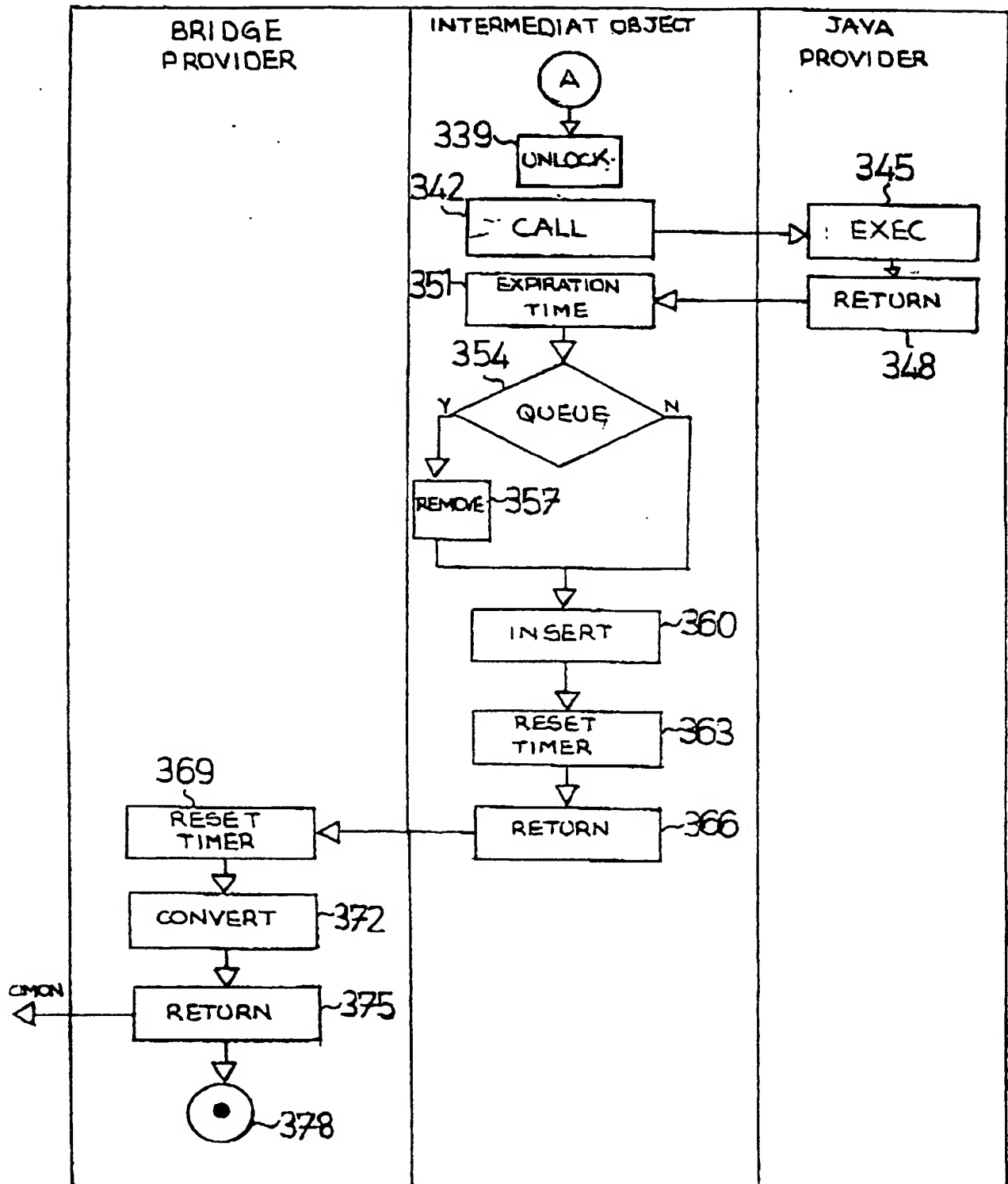


Fig.3b

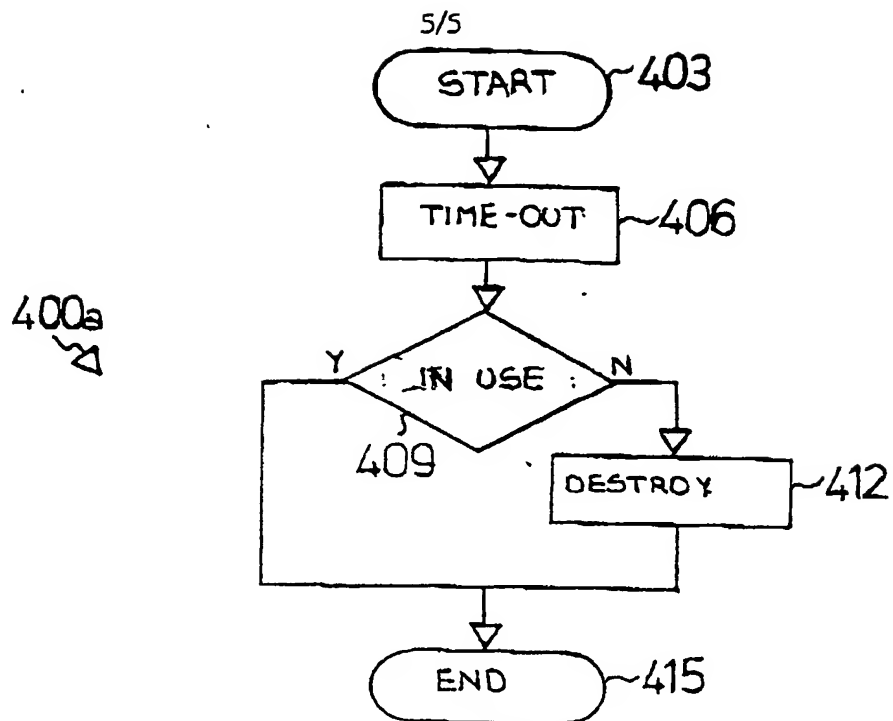


Fig. 4a

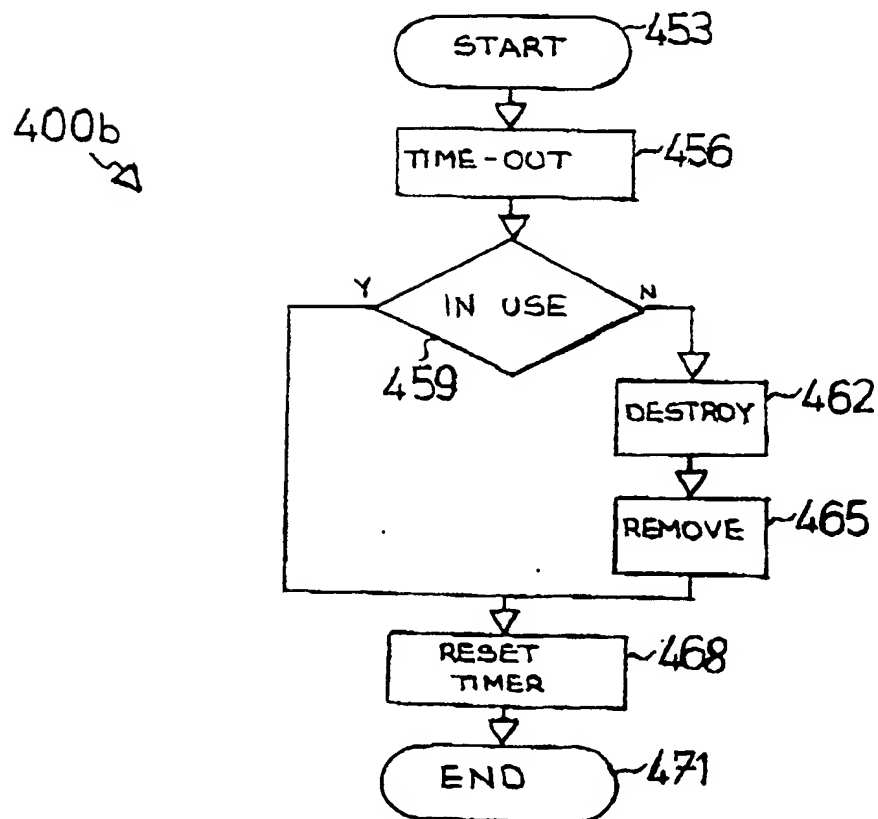


Fig. 4b

